Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Программное игровое средство «job Searcher telegram bot»

|  |  |  |
| --- | --- | --- |
| Студент |  | В. А. Довгулёв |
| Руководитель |  | асс. Е. В. Мельникова |

Минск 2020

СОДЕРЖАНИЕ

[Введение 5](#_Toc513665878)

[1 Анализ предметной области 7](#_Toc513665879)

[1.1 Обзор аналогов 7](#_Toc513665880)

[1.2 Постановка задачи 8](#_Toc513665881)

[2 Разработка программного средства 10](#_Toc513665882)

[2.1 Структура программы 10](#_Toc513665883)

[2.2 Подготовка к работе с API 10](#_Toc513665884)

[2.3 Формат JSON 13](#_Toc513665885)

[2.4 Технологии опроса сервера на наличие новых сообщений 1](#_Toc513665888)4

[2.5 Взаимодействие с Telegram Bot API](#_Toc513665888) 16

[2.6 Логика работы приложения](#_Toc513665889) 21

2.7 Архитектра работы команд приложения21

2.8 Разработка дерева меню пользователя22

2.9 Работа с картами и геокодированием22

3 Руководство пользователя24

[Заключение 28](#_Toc513665894)

[Список использованных источников 29](#_Toc513665895)

[Приложение А. Исходный код программы 30](#_Toc513665896)

ВВЕДЕНИЕ

Компьютерные сети являются средством передачи информации на некоторое расстояние с применением методов кодирования и мультиплексирования данных, которые получили развитие в разных телекоммуникационных системах.

Первые компьютеры 1950-х годов – громоздкие и дорогие, иногда занимавшие целые здания, – не предназначались для интерактивной работы пользователей, а применялись в режиме пакетной обработки. Системы пакетной обработки чаще всего были построены на базе мощного компьютера повышенной надежности универсального назначения – мэйнфрейма.

В начале 1960-х гг., с удешевлением процессоров, стали развиваться интерактивные многотерминальные системы разделения времени, в которых каждому пользователю предоставлялся отдельный терминал, который давал возможность взаимодействовать с компьютером. Мощность компьютера определяла число одновременно работающих пользователей с компьютером. Терминалы уже могли распределяться по всему предприятию, а не только в пределах вычислительного центра. При этом вычислительная мощность была еще полностью централизованной, но функции ввода и вывода данных уже стали распределенными. Многотерминальные централизованные системы такого рода внешне похожи на локальные вычислительные сети, в которых пользователь получает доступ к общим файлам и периферийным устройствам, запускает требуемую программу в любой момент времени и почти сразу же получает результат.

Назрела потребность соединения компьютеров, которые находились на большом расстоянии один от другого. Проблему начали решать через обеспечение доступа к компьютеру с терминалов, которые удалены от него на сотни километров. Терминалы соединяли с компьютерами с помощью телефонных сетей через модемы. Подобные сети давали возможность пользователям получить удаленный доступ к разделяемым ресурсам, которые находились на нескольких мощных суперкомпьютерах. Позже появились системы, в которых были реализованы удаленные соединения типа компьютер-компьютер. С помощью компьютера стал возможен обмен данными в автоматическом режиме, который является базовым признаком любой вычислительной сети. С помощью подобного механизма в первых сетях реализовывались службы обмена файлами, работа электронной почты. обеспечение синхронизации баз данных и другие, традиционные на сегодняшний день сетевые службы. Хронологически первыми стали глобальные сети (Wide Area Network, WAN) – сети, которые объединяют территориально разрозненные компьютеры, расположенные даже в различных городах и странах.

Применение интернета стало расширяться и вышло за корпоративную среду. Так, например, всемирная сеть является способом массового общения людей, которое происходит с использованием различных форумов, блогов, социальных сетей и мессенджеров.

Система мгновенного обмена сообщениями, Система обмена мгновенными сообщениями, Мессенджер (англ. Instant messaging, IM) — службы мгновенных сообщений (Instant Messaging Service, IMS), программы онлайн-консультанты (OnlineSaler) и программы-клиенты (Instant Messenger, IM) для обмена сообщениями в реальном времени через Интернет. Взаимодействие в пределах мессенджера не ограничивается общением пользователь-пользователь. Набирают также популярность и чат-боты, под которыми понимается специальная программа, выполняющая автоматические действия в ответ на действия пользователя. Взаимодействие с чат-ботом осуществляется посредством специальных команд.

Вместе с развитием мессенджера Telegram чат-боты обрели популярность. В данном мессенджере можно найти широкое множетсво различных ботов, которые выступают в качестве помощников в различных сферах. Например, боты используются для удобной подачи прогноза погоды пользователю, для предоставления расписания движения машрутного транспортного средства. Также боты могут использоваться в бизнес среде. К такому типу относятся боты, с помощью которых, например, можно оформить доставку какого-либо продукта.

Целью данного проекта является разработка чат-бота для мессенджера Telegram, позволяющего осуществить поиск открытых вакансий по определенным фильтрам, а также предоставляющий набор действий для взаимодействия с работодателем.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ
   1. Обзор аналогов

Ресурс hh.ru является одним из самых известных сайтов по поиску работы в интернете, поэтому не удивительно, что у них появился свой Telegram бот. «Helperhh\_bot» позволяет искать вакансии на сайте [hh.ru](http://www.hh.ru).

У данного бота очень приятное взаимодействие с пользователем благодаря интерактивной клавиатуре. Также в качестве плюсов можно отметить огромнейшую базу доступных вакансий, а также возможность подписки на интересующие нас запросы. Еще одним из достоинств является возможность обратной связи. Пример взаимодействия с данным Telegram ботом представлен на рисунке 1.1.

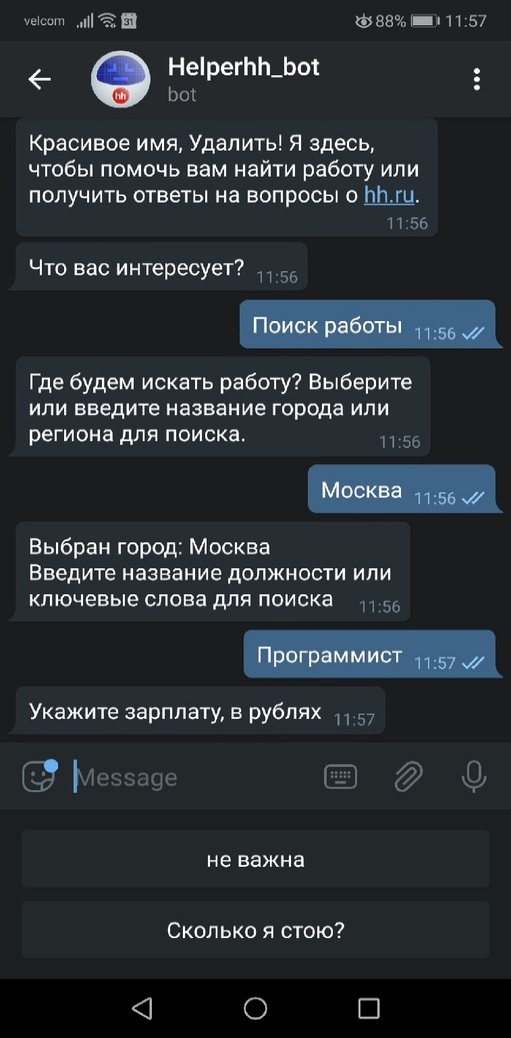
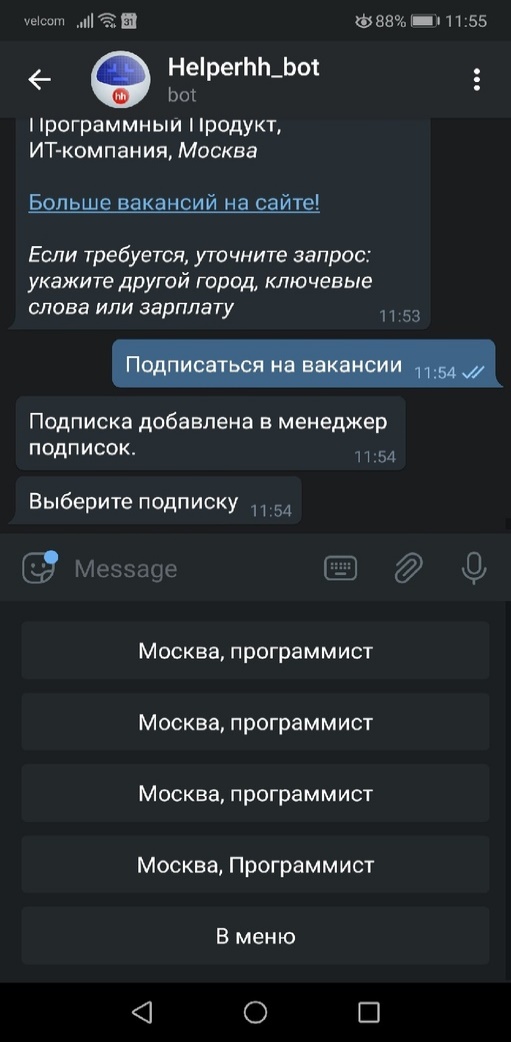
 

Рисунок 1.1 – Пример взаимодействия с Telegram ботом «Helperhh\_bot»

В ходе анализа аналога был отмечен ряд недостатков. Во-первых, интерактивная клавиатура имеет слишком большой размер, из-за чего занимает огромную часть экрана и делает неудобным просмотр сообщений. Также в параметрах поиска нет никаких других фильтров, кроме города, ключевых слов и заработной платы. Из-за постоянного смена клавиатуры и ее постоянных исчезновений и появлений постоянно происходит случайное нажатие на разные кнопки, после чего приходится возвращаться назад или начинать все сначала.

Еще одним аналогом является бот «StayaBot». Также большим плюсом является то, что бот просит подождать пользователя, так как в других ботах постоянно существует проблема, когда ты не понимаешь: бот завершил работу или загружает результаты запроса.

В данном аналоге полно недостатков. Существенным недостатком является то, что бот осуществляет подписку, но не предоставляет доступные вакансии сразу. Еще как минус стоит отметить отсутствие каких-либо фильтров, кроме выбора области. На рисунке 1.2 представлен вариант взаимодействия с данным ботом.

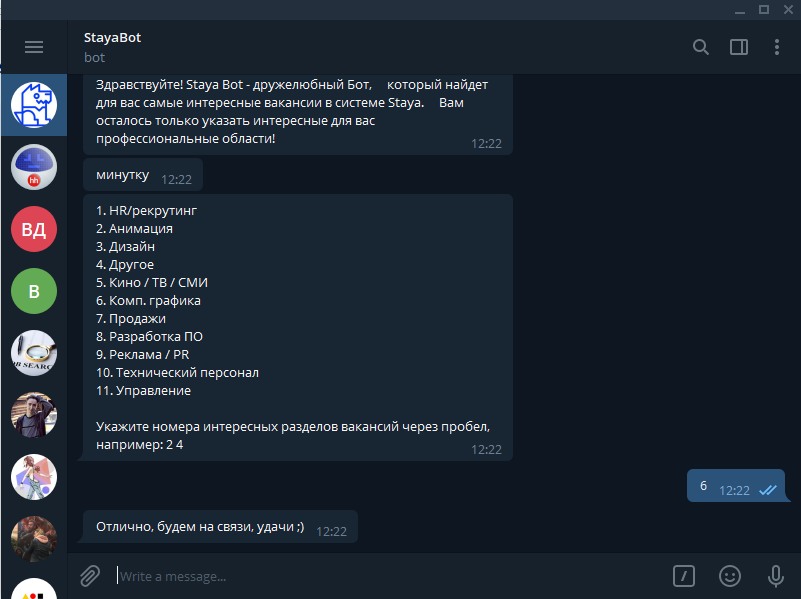


Рисунок 1.2 – Пример взаимодействия с ботом «StayaBot»

* 1. Постановка задачи

Анализируя достоинства и недостатки программ аналогов, можно определить основные требования к данному курсовому проекту, которые совмещали бы достоинства рассмотренных программ и исправляли их недостатки.

В результате сравнения аналогов программного средства и анализа предметной области в данном курсовом проекте поставлены следующие задачи:

* разработать:

а) программное средство «Telegram Bot»;

б) систему обработки ввода пользователя;

в) систему отправки запросов и получения ответов с помощью протокола HTTP;

г) алгоритм обработки JSON-ответов;

д) систему хранения и редактирования данных пользователя;

е) систему для локализации интерфейса;

ё) систему авторизации пользователя на ресурсе superjob.ru

* реализовать:

а) взаимодействие с Telegram Bot API;

б) взаимодействие с API Superjob.ru;

* создать:

а) командный интерфейс;

б) меню на основе «Inline Keyboards»;

* визуализировать данные пользователя.

Для разработки программного средства будет использоваться язык программирования Java и среда разработки Intellij IDEA 2020.1.1.

1. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА
2. 1. Структура программы

В данном приложении используется 11 пакетов, которые содержат необходимые для работы программы модули и интерфейсы:

– bot – пакет, содержащий функцию подключения бота к Telegram Bot API;

– responses – пакет, в котором находится вся логика бота касательно взаимодействия с пользователем и определения команд;

– commands – пакет, содержащий модули, в которых находится логика соответствующей команды бота;

– network – пакет, отвечающий за отправку GET/POST запросов к API для получения ботом необходимых данных;

– mappers – пакет, отвечающий за преобразование JSON-ответов и загрузку данных из них в соответствующие логике приложения коллекции;

– locale – пакет, в котором содержится модуль LocalServiceImpl для реализации смены языка пользовательского интерфейса в приложении;

– user – пакет, в котором содержатся модули, содержащие необходимые модули для хранения данных о чатах и пользователях в них;

– vacancies – пакет, в котором содержатся модули соответствующие данным о вакансиях отправляемые API;

– authorization – пакет, в котором содержатся модули, реализующие логику авторизации пользователя;

– filters – пакет, содержащий модули, в которых реализована логика работы фильтров поиска работы в приложении;

– errors – пакет, содержащий модуль для обработки ошибок ввода пользователя.

* 1. Подготовка к работе с API

Приложение, реализующие бота для Telegram, представляет из себя сервер, взаимодействующий с сервером Telegram, используя созданный для этого Bot API, т.е. программный интерфейс приложения, с помощью которого после отправки сообщения на URL с параметрами в соответствии с интерфейсом можно получать ответ в стандартизированном формате JSON/XML. Взаимодействие происходит по протоколу HTTPS.

Для работы с Telegram Bot API была использована библиотека TelegramBots для языка программирования Java, имеющая лицензию MIT. Библиотека добавляет новый уровень абстракции при взаимодействии с API, что облегчается и ускоряет разработку бота.

Как правило, для использования API какого-то сервиса, необходим специальный ключ, или токен, для идентификации клиентов в системе. На токене базируется все взаимодействие с сервисом мессенджера, так что тот, кто знает токен, фактически владеет ботом. Для управления и получения доступа к ботам Telegram предоставляет бота BotFather. С помощью него можно создать нового бота. Для этого нужно отправить боту команду «/newbot». Далее указывается имя, отображаемое пользователям при взаимодействии с вашим ботом. После этого выбирается имя пользователя бота, оно обязательно должно заканчиваться на «bot». В конце вам будет выслан токен для зарегистрированного бота. Пример полученного токена показан на рисунке 2.1.

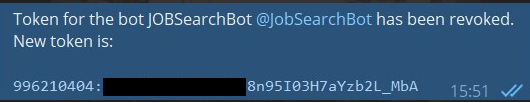
****

Рисунок 2.1 – Полученный токена для бота

Также для получения вакансий необходимо реализовать взаимодействие с API интегрированной информационной системы. Для того, чтобы использовать весь функционал Superjob API необходимо зарегистрировать приложение на сайте api.superjob.ru. Далее, получив при регистрации секретный ключ (Secret key) вашего приложения, необходимо его предавать в заголовке запроса X-Api-App-Id. Также необходимо произвести авторизацию перед отправлением некоторых запросов, чтобы получить Access Token. Согласно документации Superjob API авторизация производится по протоколу OAuth v.2.0. Данный протокол имеет следующую последовательность действий. Сначала производится запрос на пользовательскую авторизацию. После авторизации перенаправление на указанную страницу с параметром code. Для получения Access Token производится запрос на соответствующий метод API. В случае истечения срока годности Access Token используется Refresh Token. Для всех методов API, требующих авторизацию, подставлять Access Token в заголовок Authorization. Пример заголовка запроса показан на рисунке 2.2.

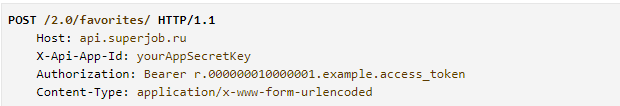


Рисунок 2.2 – Пример заголовка запроса к API

Существует и другой метод авторизации – авторизация по паролю. Данный метод необходимо использовать с передачей в запросе следующих параметров: login, password, client\_id, client\_secret, hr. Пример такого запроса представлен на рисунке 2.3. Для предотвращения подбора пароля, в случае обращения к методу с одним и тем же логином, допускается не более 10 ошибок в час. В случае превышения этого лимита, выдаётся ошибка авторизации с кодом 429. Данный метод и был выбран для авторизации в приложении.



Рисунок 2.3 – Пример запроса авторизации по паролю

Однако не всегда необходимо передавать свой секретный ключ в заголовке. Всё зависит от того, требуется ли для данного запроса аутентификация. На рисунке 2.4 показан ответ API на запрос в браузере.

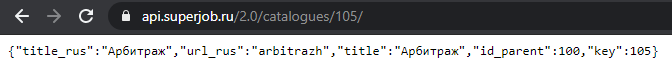
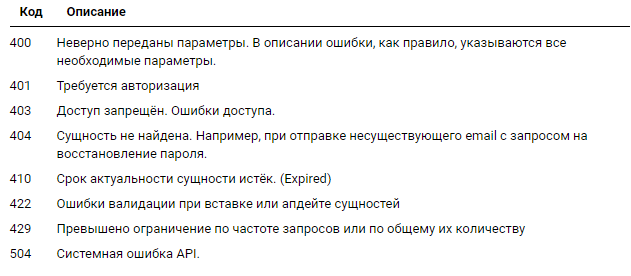


Рисунок 2.4 – JSON-ответ с api.superjob.ru

В данном API также присутствует ограничение по частоте запросов – 120 запросов в минуту с одного IP-адреса. В случае превышения нормы сервер будет выдавать ошибку с кодом 503.

Коды ошибок, возвращаемые API, совпадают с HTTP-кодами. Описание оишбок представлено на рисунке 2.5.



Данный API содержит методы как для поиска вакансий (с точки зрения соискателя), так и для их создания и отслеживания (с точки зрения работодателя).

* 1. Формат JSON

JSON (JavaScript Object Notation) - простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. Он основан на подмножестве языка программирования JavaScript и хранит структурированную информацию и в основном используется для передачи данных между сервером и клиентом.

Файл JSON представляет собой более простую и лёгкую альтернативу расширению с аналогичными функциями XML (eXtensible Markup Language). Во многих случаях, XML очень схож с JSON, но не такой лаконичный, таким образом становясь длиннее и потребляя больше времени для чтения и записи.

Основными элементами объекта JSON являются ключи, которые всегда являются строками, и значения определенных форматов. Объект заключается в фигурные скобки, далее через запятую перечисляются пары ключ/значение с двоеточием посередине. Для иллюстрации JSON-объекта на рисунке 2.5 приведен пример такового, полученного при взаимодействии с Superjob API.

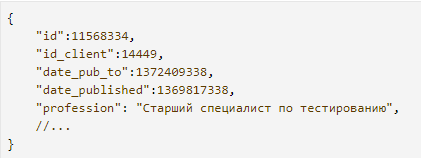


Рисунок 2.5 – Пример JSON

Для работы с JSON-форматом в языках программирования существуют многочисленные библиотеки. Обработка объектов производится обращением к полям, которые должны быть заранее известны.

При написании парсера JSON, была использована библиотека Jackson. В частности модуль ObjectMapper. Парсинг происходил двумя способами: частично – с применением метода get(), который по имени поля возвращал его значение и полный – с применением метода readerFor(). Однако во втором случае возникла одна проблема – несовпадение названий полей в модуле и JSON. Проблема была решена добавлением аннотаций @JsonProperty(имя\_поля) для каждого поля, где имя\_поля это название непосредственно в JSON.

* 1. Технологии опроса сервера на наличие новых сообщений

Для обработки сообщений пользователей бот должен своевременно их получать. Сообщения приходят на сервера Telegram и могут быть переданы боту несколькими способами. Для начала рассмотрим весь спектр существующих технологий.

На данный момент существуют следующие механизмы опроса сервера на появление новых событий. Наиболее популярными являются следующие:

* Polling – периодический опрос сервера;
* Comet (Long polling) – polling с долгоживущими запросами;
* Server Push – бесконечный запрос;
* Webhook («крючок») – сервер сам уведомляет об изменениях.

Bot API позволяет организовать получение обновлений по технологиям Long Polling, а также Webhook. Данные технологии позволяют использовать протокол HTTP в системах реального масштаба времени, когда система должна реагировать на события во внешней по отношению к себе среде в рамках требуемых временных ограничений. Собственно система моментального ответа на сообщения таковой и является.

Long Polling («длинный опрос») – технология, в соответствии с которой клиент опрашивает сервер как и при обычной технологии Polling, но не требует от сервера моментального ответа. При обычном опросе не гарантируется моментальное получение обновления, т.к. клиент посылает запрос в определенные моменты и не знает, пришло ли новое сообщение. Сервер загружается большим количеством запросов, которые, как правило, в большинстве не информативны. Схема работы технологии Polling показана на рисунке 2.6. К плюсам данной технологии можно отнести



Рисунок 2.6 – Технология Polling

В технологии же «длинного опроса» задается определенный тайм-аут, т.е. время ожидания. Если никакого события на сервере не происходит, ответ во время ожидания не отправляется. Когда тайм-аут истекает, сервер оповещает о состоянии системы. Если в период времени ожидания происходит какое-то событие, сервер моментально оповестит клиента о нем. При обычном опросе подавляющее число ответов содержит «Нет событий». Поэтому технология Long Polling позволяет сократить число опросов, которые не несут пользу, и снижает нагрузку на сервер. На рисунке 2.7, который расположен ниже, показана схема работы технологии «длинного опроса».



Рисунок 2.7 – Технология Long Polling

Выбор данной технологии при разработке бота для Telegram мотивируется простотой реализации, а также пропускной способностью. Если бот не является популярным, то Long Polling самое подходящее решение в соотношении «сложность внедрения – производительность».

В технологии Webhook клиент бота «подвешивается на крючок» со стороны сервера. Т.е. теперь клиенту не нужно опрашивать сервер о том, пришли ли для него новые сообщения. Обязанность уведомлять клиент ложиться на сервер. Webhook можно назвать «функцией обратного вызова HTTP, заданной пользователем». Имеется в виду, что изначально клиент бота должен проинициализировать свой URL-адрес и токен на сервере с помощью Telegram API. Далее все уведомления будут высылаться на переданный URL.

Webhook, как и Long Polling, позволяет получать уведомления моментально, однако Webhook снижает нагрузку на сервер Telegram до минимума. Все ответы сервера будут являться информативными, содержащими данные о событии. Хорошим тоном является разделение инфраструктуры на клиент бота, сервер уведомлений и API для ответных сообщений, хотя по факту два последних являются частью Telegram Bot API. Схема с вышеизложенным определением действующих лиц находится далее на рисунке 2.8.

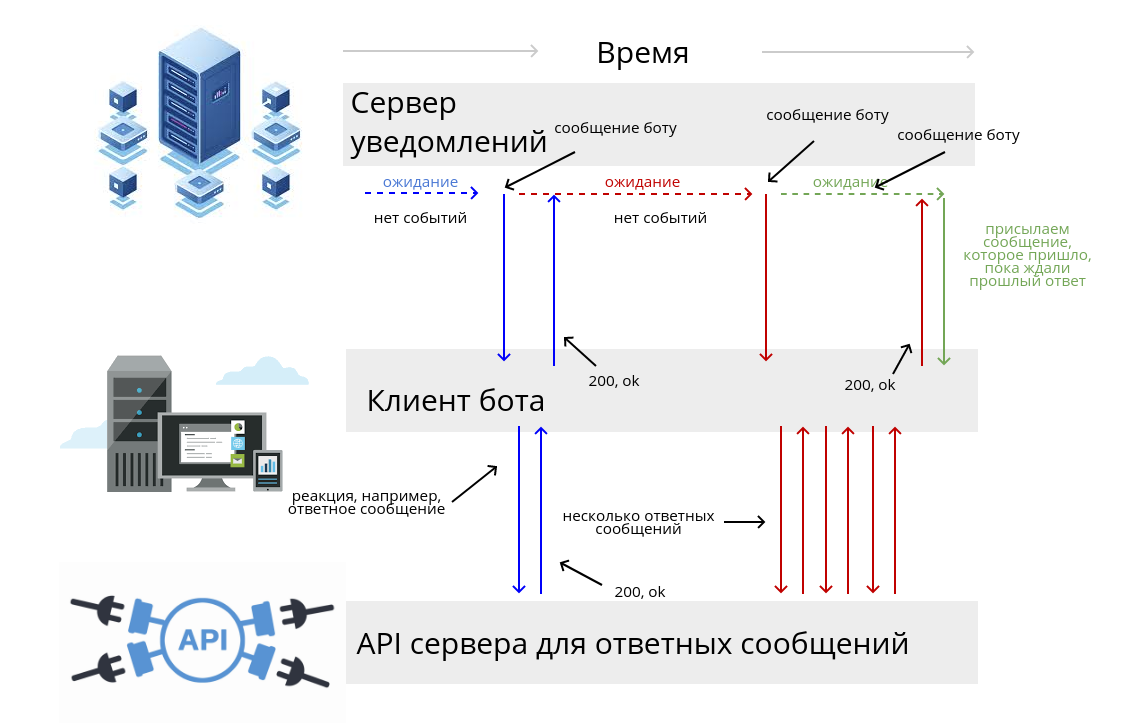


Рисунок 2.8 – Технология Webhook

Как видно из схемы работы технологии Webhook, клиент бота фактически становиться сервером, для установки «крючка» необходим внешний адрес в сети. Для этого нужно получить либо IP-адрес, либо доменное имя. Но это требуется дополнительных денежных затрат, либо можно бесплатно получить динамический адрес, однако его придется периодически обновлять на сервере Telegram. Поэтому использование Webhook предпочтительно для популярных ботов с большой нагрузкой и финансовой поддержкой.

В данном курсовом проекте реализован бот по технологии Long Polling, но рассматривается возможность перехода на Webhook.

* 1. Взаимодействие с Telegram Bot API

Взаимодействие с Telegram Bot API организуется с использованием протоколов различных уровней, которые будут рассмотрены в порядке следования в сетевой модели OSI (рисунок 2.9).

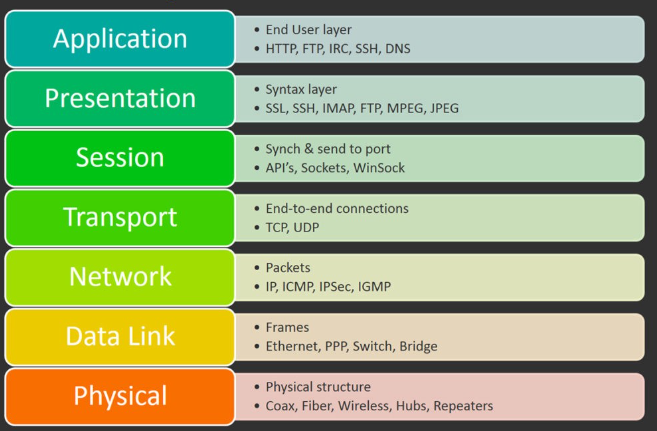


Рисунок 2.9 – Сетевая модель OSI

На верхнем уровне происходит взаимодействие по HTTP-интерфейсам.

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи данных в виде гипертекстовых документов, однако сейчас он используется для передачи любых данных. Обмен данными происходит в текстовом формате, т.к. протокол реализован поверх TCP (рисунок 2.10).

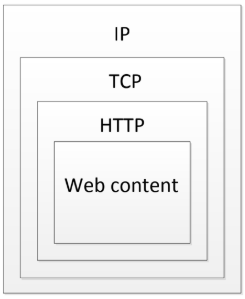


Рисунок 2.10 – Инфраструктура HTTP

Во взаимодействии по HTTP используется понятие транзакции: клиент подключается к серверу, после чего отправляет запрос, а сервер возвращает ответ на запрос. При этом структура HTTP-запроса имеется следующий вид (рисунок 2.11).

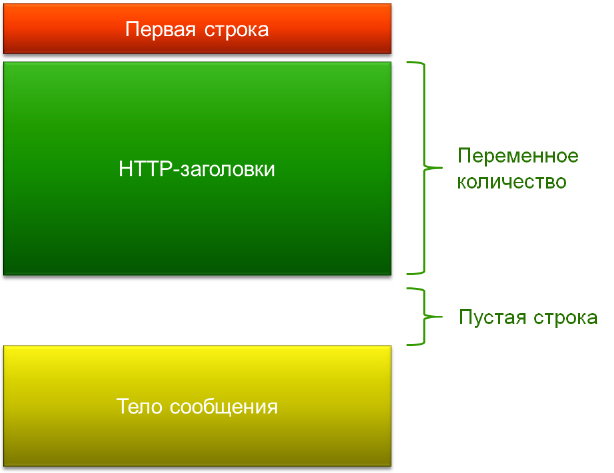


Рисунок 2.11 – Структура HTTP-запроса

В строке первой строке (строке запроса) указывается метод, путь к ресурсу, а также версия протокола. Bot API поддерживает методы GET и POST, однако это далеко не все методы, предоставляемые протоколом HTTP. Метод представляет из себя обозначение типа действия. Так, например, метод GET означает, что клиент запрашивает содержимое ресурса, к которому указан путь. Под ресурсами в Telegram Bot API скрываются вызовы удаленных процедур, поэтому API можно назвать своего рода веб-сервисом. В заголовке запроса и ответа перечисляются различные настройки в формате «имя:значение», например браузер может указать формат данных, с которыми он может работать. В случае работы с Telegram Bot API, в заголовке ответа указывается, что данные отправляются в формате JSON. С использованием метода GET реализованы такие функции, как getMe, которая возвращает информацию о боте, или, например, getUpdates, которая используется для ботов, использующих технологию Long Polling. Метод POST означает, что клиент отправляет данные на сервер, тогда в теле запроса указываются данные, следует отметить, что при использовании метода GET тело запроса пустое. С помощью метода POST на сервер Telegram, например, отправляется сообщение бота, адресованное пользователю. Для этого используются различные функции API, которые зависят от типа отправляемых данных: sendMessage, sendPhoto, sendAudio и так далее.

Протокол HTTP является «stateless» протоколом, т.е. протоколом без сохранения состояния. Это означает, что в нем не реализовано логическое понятие сессии. Данный факт позволяет реализовывать распределенные системы с балансировкой нагрузки, в которых балансировщик может отправлять запросы клиента не напрямую к серверу, а к серверу-кэшу, что снижает нагрузку на основной сервер. На сервере-кэше хранятся ответы на идемпотентные запросы, т.е. на те, результат которых одинаков при одинаковых параметрах запроса. Но при этом, серверу все-таки необходимы данные сессии для авторизации клиентов, ведь доступ к некоторым данным может быть ограничен. Поэтому клиент передает свои данные для авторизации внутри заголовков HTTP.

Для авторизации необходимо пройти процедуру аутентификации, т.е. процедуру определения личности пользователя. Это осуществлялось передачей последовательности «имя:пароль», закодированной по стандарту Base64, или хеш-значения, полученного по алгоритму MD5, в заголовке HTTP.

Можно сказать, что первый способ не предоставлял никакой защиты личных данных пользователя при прослушивании промежуточных узлов. Однако при использовании второго способа никто не мог гарантировать, что вместо того, чтобы получить данные для авторизации, злоумышленник просто не начнет читать тело HTTP-ответов, которые представляют из себя те самые защищенные данные. Таким образом, существовавшие способы авторизации не предоставляли никакой защиты трафика от нежелательного просмотра, что было очень критично при организации работы, например, банковской системы в сети Интернет или приложения по обмену сообщениями, такого как Telegram.

Для защиты данных используется расширение протокола HTTP под названием HTTPS. Последний реализован поверх протокола SSL представительного уровня. Схема его работы изображена на рисунке 2.12.

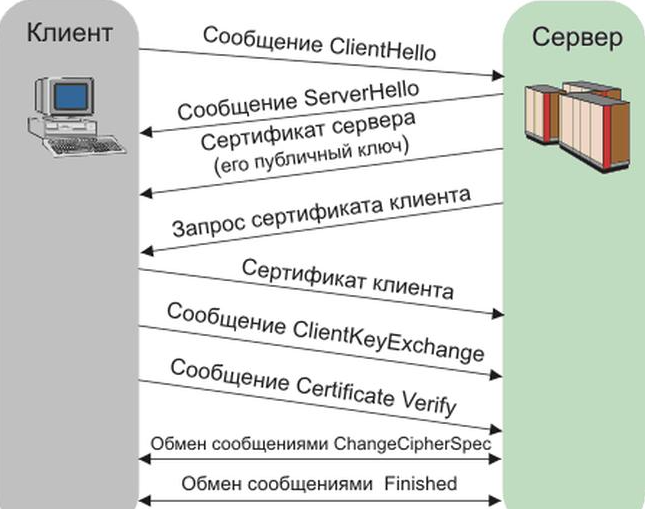


Рисунок 2.12 – Схема работы протокола SSL

Он обеспечивает шифрование трафика, которое происходит на уровне сокета, т.е. на уровне установки соединения. Все данные, пересылаемые между сервером и клиентом, шифруются. При чем можно не сразу устанавливать соединение защищенным, имеется возможность перейти из незащищенного режима в защищенный во время работы. Для организации SSL-соединения сервер выдает сертификат, представляющий из себя цифровую подпись, чтобы клиент мог проверить подлинность сервера. Далее клиент отправляет серверу симметричный ключ, который зашифрован открытым ключом, сгенерированным на основе сертификата подлинности сервера. После чего все переданные данные шифруются симметричным ключом, который сервер получит путем дешифрирования сообщения от клиента с помощью закрытого ключа. Серверу же для проверки подлинности клиента достаточно провести аутентификацию через заголовки HTTP, которые уже зашифрованы. Поэтому сейчас даже не используются способы, характерные для HTTP, которые описаны ранее, а передается так называемый токен в открытом в виде, т.к. само соединение уже зашифровано. Ниже приведен пример URL, по которому происходит обращение к Telegram Bot API, токен приводится в явном виде, а в строке запроса указывается метод, предоставляемый API.

https://api.telegram.org/bot<token>/METHOD\_NAME

Все запросы и ответы протокола HTTP пересылаются поверх протокола транспортного уровня TCP. Пакеты маршрутизируются по протоколу IP сетевого уровня и преобразуются по протоколам нижестоящих уровней.

* 1. Логика работы приложения

Все сообщения, отправленные пользователем боту, сохраняются на сервере Telegram. С помощью вызова функции API getUpdates бот начинает делать «долгий опрос» сервера, поэтому своевременно получает новые сообщения от пользователей и приступает к их обработке. В справке к боту жестко стандартизирован формат сообщения пользователя, что способствует быстрой программной обработке. Соответственно определяется, какую функцию бота вызывает пользователь. Из запроса выделяются параметры. Далее выполняется HTTP-запрос, согласно документации Superjob API. Полученный JSON-ответ приводится к человекочитаемому виду и отправляется в ответ пользователю на сервер Telegram, который во всем взаимодействии выполняет роль прокси-сервера и пересылает сообщение дальше к пользователю.

* 1. Архитектура команд приложения

Самой сложной частью логики приложения является обработка команд пользователя. Для того, чтобы обработать команду необходимо знать текущее состояние (где с точки зрения программы находится пользователь), имя команды, id пользователя и некоторые другие специфические данные. Изначально была попытка сделать один класс, в котором будут содержаться статические методы обработки каждой команды. Благодаря этому можно было бы легко передавать необходимые данные в качестве параметров в каждый метод. Определение метода происходило бы по соответствию имени команды и её обработчика. Однако такой подход привёл к плохим последствиям, а именно нечитаемость кода и его большое количество в одном модуле. И это при том, что к тому моменту была реализована только половина команд. В связи с этим было решено изменить подход. В новом варианте реализации присутствует абстрактный класс Command, от которого наследуются все команды. Теперь у них у всех есть один метод execute(), который возвращает результат её выполнения. А создание объектов происходит после определения типа команды. Данный подход позволил разделить код между модулями и упростил добавление новых и поддержку старых команд.

* 1. Разработка дерева меню пользователя

Для пользователя приложения очень важен понятный и лаконичный интерфейс. Именно поэтому не стоит создавать лишний функционал в таком приложении как чат-бот. Это будет усложнять работу с ним. Был выбран следующий подход – каждая команда должна быть максимально эффективной и простой. Дерево команд не должно быть глубоким и запутанным. В результате было выделено следующее: главное меню, меню фильтров, меню просмотр подробностей о вакансии, меню авторизации. В главном меню находятся команды для смены языка, переход в профиль, настройка фильтров и авторизация. В меню фильтров находятся сами фильтры, команда их очищения и команда поиска. В меню подробной информации о вакансии отображается дополнительная информация о вакансии и команда добавления в избранное. В меню авторизации можно зайти или выйти из своего аккаунта. Пример отображения одного из меню представлен на рисунке 2.13.

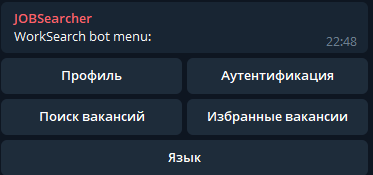


Рисунок 2.13 – Главное меню бота

* 1. Работа с картами и геокодированием

Для того, чтобы показывать пользователю местоположение определённых мест на карте необходимо получить координаты данного места. Однако не всегда является возможным получить их напрямую из данных, которые нам возвращает Superjob API. Поэтому был использован API геокодера от яндекс, который по адресу, получаемому из ответа Superjob API, получал координаты. Эти координаты передавались в метод SendLocation() и Telegram отображал карты в сообщении, где можно было увидеть местоположение вакансии. Результат работы представлен на рисунке 2.14.

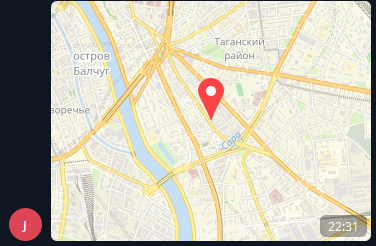


Рисунок 2.14 – Пример отображения локации

1. руководство пользователя

Для того, чтобы попасть в главное меню пользователь дожен отправить боту команду «/menu», которая выводит кнопки функционала приложения. Результат работы команды представлен на рисунке 4.1.

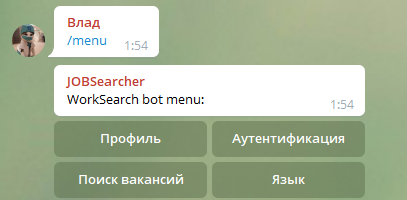


Рисунок 4.1 – Команда «/menu»

Для того, чтобы авторизоваться пользователь должен выбрать пункт главного меню аутентификация, а затем ввести команду «/login» и передать туда логин и пароль. Использование данной команды изображено на рисунке 4.2.

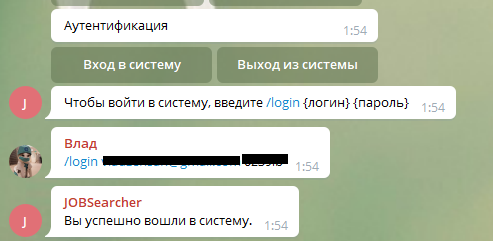


Рисунок 4.2 – Команда «/login»

Данный бот также предусматривает смену языка. Это происходит с использованием команды «/lang». На рисунке 4.3 показан вызов этой команды и результаты её работы.

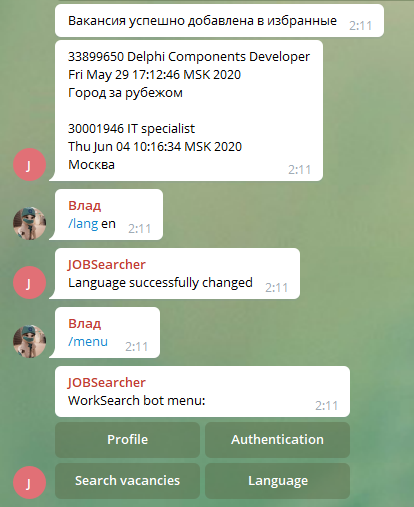


Рисунок 4.3 – Смена языка

Для установки фильтров и поиска вакансий по ним нужно перейти в пункт меню поиск вакансий. Интерфейс команды приведен далее на рисунке 4.4.

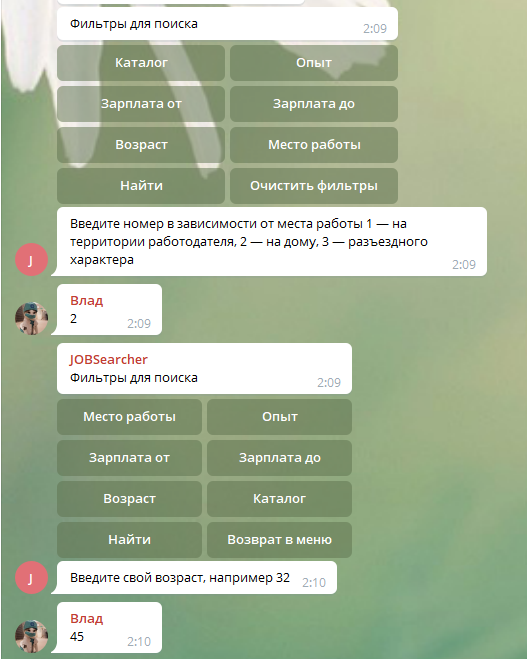


Рисунок 4.4 – Пример использования фильтров

Список вакансий выводится по нажатию на кнопку поиск. Так как вакансий может быть достаточно много, то их лучше выводить небольшими частями. При этом для навигации среди всех вакансий используются кнопки вперёд и назад. Примеры получения списка вакансий показаны на рисунке 4.5.

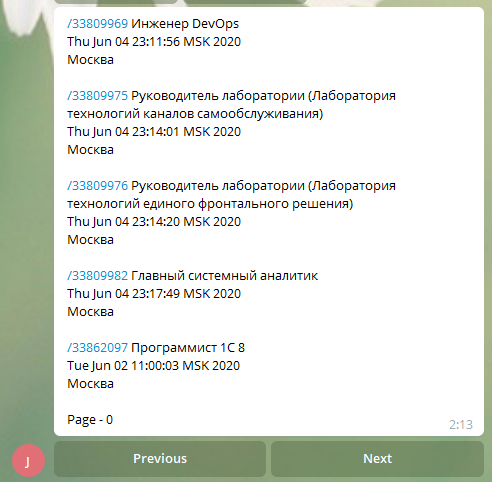


Рисунок 4.5 – Получение списка вакансий

Получение подробной информации о вакансии, а также добавление вакансии в избранные происходит по нажатию на неё id в списке вакансий. Прмер работы представлен на рисунке 4.6.

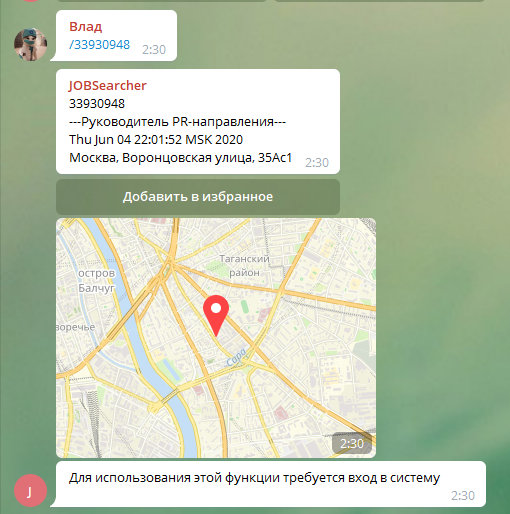


Рисунок 4.6 – Подробная информация о вакансии

После того, как пользователь авторизовался он может просмотреть список избранных вакансий. Пример работы показан на рисунке 4.7.

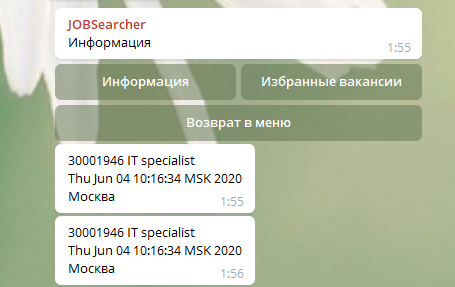


Рисунок 4.7 – Список избранных вакансий

ЗАКЛЮЧЕНИЕ

Важность компьютерных технологий растет с каждым днем. Они позволяют существенно упросить разного рода рутинную деятельность, упростить доступ к информации и взаимодействие людей.

В результате данного курсового проектирования было разработано программное средство «Telegram Bot», которое позволяет получать список вакансий по сформированным с помощью фильтров запросам, а так же создавать своё резюме. Согласно поставленным задачам, в данном приложении были реализованы следующие функции:

* авторизация пользователя;
* получение списка вакансий;
* получение подробной информации о вакансии;
* добавление в избранное;
* отображение:

а) командного интерфейса;

б) кнопок для быстрого доступа;

* отображение местонахождения работы;
* поддержка двух языков.

Успешное выполнение задач было бы невозможным без детального изучения возможностей программного интерфейса приложения Telegram, а в частности Bot API, протокола HTTP, документации Superjob API, особенностей средства разработки Intellij IDEA и распределённой системы управления версиями Git.

Существует ряд направлений дальнейшего развития разработанного приложения. Одним из таких направлений является упаковка всего приложения в Docker-контейнер и разворачивание его на облачной PaaS-платформе Heroku, тем самым автоматизировав работу приложения. Кроме того, будет рассмотрена возможность получения IP-адреса, на который можно будет установить Webhook, тем самым увеличив скорость ответа бота.

Данное приложение обладает удобным и лаконичным интерфейсом, средствами для быстрого получения ответа на запрос пользователя и призвано облегчить доступ к необходимой информации. Кроме того, оно позволяет сохранять некоторые данные для последующего использования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Telegram Bot API [Электронный ресурс]. – Электронные данные. – Режим доступа[: https://core.telegram.org/bots/api](file:///C:\Users\marta\Downloads\:%20https:\core.telegram.org\bots\api).
2. Java Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа[: <https://docs.oracle.com/en/java/>](file:///C:\Users\marta\Downloads\:%20https:\core.telegram.org\bots\api).
3. Шилдт, Г. Java. Полное руководство / Г. Шилдт – Москва : изд. «Вильямс», 2015. – 1377 с
4. Э. Таненбаум, Д. Уэзерпол. Компьютерные сети. Учебное издание. – Питер, 2006.
5. Д. А. Сурков, Е. В. Мельникова. Сети ЭВМ. Учебное издание. – БГУИР, 2004.
6. В. Г. Олифер, Н. А. Олифер. Компьютерные сети, принципы, протоколы, технологии учеб. Учебник для вузов. – Питер, 2015.
7. Superjob API [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://api.superjob.ru/>.
8. API карт [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://tech.yandex.ru/maps/geocoder/>.

ПРИЛОЖЕНИЕ А

Исходный код программы

ResponseServiceImpl.java:

package responses;

import commands.Command;

import commands.CommandCreator;

import commands.CommandEnum;

import commands.auth.AuthCommand;

import commands.auth.LoginCommand;

import commands.auth.LogoutCommand;

import commands.profile.FavoritesCommand;

import commands.profile.LanguageCommand;

import commands.profile.ProfileCommand;

import commands.profile.ProfileInfoCommand;

import commands.search.\*;

import commands.utility.BackMenuCommand;

import commands.utility.MenuCommand;

import commands.utility.NextPageCommand;

import commands.utility.PrevPageCommand;

import org.telegram.telegrambots.meta.api.objects.Update;

import user.ArrayListUserDao;

import user.User;

import user.UserDao;

import java.util.Arrays;

import java.util.HashMap;

import java.util.Map;

public class ResponseServiceImpl implements ResponseService {

private UserDao userDao = ArrayListUserDao.getInstance();

private Map<CommandEnum, CommandCreator> commandCreator = getCommands();

public Response getResponse(Update update) {

String request = "undefined";

long chatId = 0L;

if (update.hasMessage()) {

request = update.getMessage().getText();

chatId = update.getMessage().getChatId();

} else if (update.hasCallbackQuery()) {

request = update.getCallbackQuery().getData();

chatId = update.getCallbackQuery().getMessage().getChatId();

}

validatePresentUser(chatId);

String[] words = request.split("\\s+");

CommandEnum commandEnum = defineCommand(words[0]);

System.out.println(commandEnum.getCommand());

CommandCreator creator = commandCreator.get(commandEnum);

User user = userDao.getUserById(chatId);

user.setCurrentUpdate(update);

Command command = creator.create(request, user);

System.out.println("shit");

return command.execute();

}

private void validatePresentUser(long chatId) {

if (!userDao.containsChatId(chatId)) {

userDao.addUser(new User(chatId));

}

}

private CommandEnum defineCommand(String request) {

return Arrays.stream(CommandEnum.values())

.filter(command -> command.getCommand().equals(request.trim().toLowerCase()))

.findAny()

.orElse(CommandEnum.OTHER);

}

private Map<CommandEnum, CommandCreator> getCommands() {

Map<CommandEnum, CommandCreator> commands = new HashMap<>();

commands.put(CommandEnum.MENU, MenuCommand::new);

commands.put(CommandEnum.LANGUAGE, LanguageCommand::new);

commands.put(CommandEnum.AUTH, AuthCommand::new);

commands.put(CommandEnum.AGE, AgeCommand::new);

commands.put(CommandEnum.BACK\_MENU, BackMenuCommand::new);

commands.put(CommandEnum.CATALOGUES, CataloguesCommand::new);

commands.put(CommandEnum.EXPERIENCE, ExperienceCommand::new);

commands.put(CommandEnum.FIND, FindCommand::new);

commands.put(CommandEnum.FAVORITES, FavoritesCommand::new);

commands.put(CommandEnum.LOGIN, LoginCommand::new);

commands.put(CommandEnum.LOGOUT, LogoutCommand::new);

commands.put(CommandEnum.NEXTPAGE, NextPageCommand::new);

commands.put(CommandEnum.PREVIOUSPAGE, PrevPageCommand::new);

commands.put(CommandEnum.OTHER, OtherCommand::new);

commands.put(CommandEnum.SEARCH, SearchCommand::new);

commands.put(CommandEnum.PROFILE, ProfileCommand::new);

commands.put(CommandEnum.PROFILE\_INFO, ProfileInfoCommand::new);

commands.put(CommandEnum.PLACEOFWORK, WorkPlaceCommand::new);

commands.put(CommandEnum.SALARYFROM, SalaryFromCommand::new);

commands.put(CommandEnum.SALARYTO, SalaryToCommand::new);

commands.put(CommandEnum.CLEARFILTERS, ClearFiltersCommand::new);

commands.put(CommandEnum.ADDFAVORITES, AddFavouritesCommand::new);

return commands;

}

}

NetworkServiceImpl.java:

package network;

import authorization.AuthService;

import authorization.AuthServiceImpl;

import authorization.AuthToken;

import filters.Filter;

import mappers.EntitiesMapper;

import mappers.JsonEntitiesMapper;

import user.UserInfo;

import vacancies.Catalogue;

import vacancies.VacanciesInfo;

import vacancies.Vacancy;

import javax.security.auth.login.FailedLoginException;

import javax.ws.rs.NotAuthorizedException;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.net.HttpURLConnection;

import java.net.URL;

import java.util.List;

import java.util.Map;

import java.util.Objects;

public class NetworkServiceImpl implements NetworkService {

private final String APP\_KEY = "v3.r.132136870.4ee38e902a0d001e916d40c50ebc65a0462696ec.cd36a27c87d8e690a1884ec256b337763d26d188";

private final String APP\_ID = "1463";

private final String AUTH\_PATH = "https://api.superjob.ru/2.0/oauth2/password/";

private final String CATALOGUES\_PATH = "https://api.superjob.ru/2.0/catalogues";

private final String VACANCIES\_PATH = "https://api.superjob.ru/2.0/vacancies/?";

private final String CURRENT\_USER\_PATH = "https://api.superjob.ru/2.0/user/current";

private final String FAVORITES\_PATH = "https://api.superjob.ru/2.0/favorites/";

private final String TOKEN\_PATH = "https://api.superjob.ru/2.0/oauth2/access\_token/";

private EntitiesMapper entityMapper = new JsonEntitiesMapper();

public AuthToken getAccessToken(String login, String password) throws FailedLoginException {

URL url;

AuthToken authToken;

try {

String path = AUTH\_PATH

+ "?login=" + login

+ "&password=" + password

+ "&client\_id=" + APP\_ID

+ "&client\_secret=" + APP\_KEY;

url = new URL(path);

HttpURLConnection con = (HttpURLConnection) url.openConnection();

con.setRequestMethod("GET");

if (con.getResponseCode() == 200) {

String jsonData = readData(con);

authToken = entityMapper.extractToken(jsonData);

return authToken;

} else {

System.out.println("Error occurred");

throw new FailedLoginException();

}

} catch (IOException e) {

e.printStackTrace();

return null;

}

}

@Override

public boolean logout(long chatId) {

String token = tryGetAuthToken(chatId);

URL url;

try {

url = new URL(TOKEN\_PATH);

HttpURLConnection con = (HttpURLConnection) url.openConnection();

con.setRequestProperty("X-Api-App-Id", APP\_KEY);

con.setRequestMethod("DELETE");

con.setDoOutput(true);

writeData(con.getOutputStream(), "access\_token=" + token.substring(token.indexOf(' ') + 1));

return con.getResponseCode() == 204;

} catch (IOException e) {

e.printStackTrace();

return false;

}

}

@Override

public List<Catalogue> getCataloguesList() {

List<Catalogue> catalogues;

try {

URL url = new URL(CATALOGUES\_PATH);

HttpURLConnection con = (HttpURLConnection) url.openConnection();

con.setRequestMethod("GET");

if (con.getResponseCode() == 200) {

String jsonData = readData(con);

catalogues = entityMapper.mapCatalogues(jsonData);

return catalogues;

} else {

System.out.println("Error occurred");

}

} catch (IOException e) {

e.printStackTrace();

}

return null;

}

@Override

public List<Vacancy> getVacanciesList(Map<Filter, String> searchParameters) {

List<Vacancy> vacancies;

try {

StringBuilder path = new StringBuilder(VACANCIES\_PATH);

if (searchParameters != null) {

for (Filter parameter : searchParameters.keySet()) {

path.append(parameter.getName());

path.append("=");

path.append(searchParameters.get(parameter));

path.append("&");

}

}

HttpURLConnection con = requestGet(path.toString(), null);

if (con != null && con.getResponseCode() == 200) {

String jsonData = readData(con);

System.out.println(jsonData);

VacanciesInfo vacanciesInfo = entityMapper.mapVacanciesInfo(jsonData);

if (vacanciesInfo.getTotal() == 0) {

return null;

} else {

vacancies = new JsonEntitiesMapper().mapVacancies(vacanciesInfo.getObjects());

}

return vacancies;

} else {

System.out.println("Error occurred");

}

} catch (IOException e) {

e.printStackTrace();

}

return null;

}

@Override

public Vacancy getVacancy(long vacancyId) {

return null;

}

@Override

public List<Vacancy> getFavoriteVacancies(long chatId) {

String token = tryGetAuthToken(chatId);

List<Vacancy> vacancies;

try {

HttpURLConnection connection = requestGet(FAVORITES\_PATH, token);

if (connection != null && connection.getResponseCode() == 200) {

String json = readData(connection);

VacanciesInfo vacanciesInfo = entityMapper.mapVacanciesInfo(json);

vacancies = new JsonEntitiesMapper().mapVacancies(vacanciesInfo.getObjects());

return vacancies;

}

} catch (IOException e) {

e.printStackTrace();

}

return null;

}

@Override

public boolean addFavoriteVacancy(long chatId, long vacancyId) {

String token = tryGetAuthToken(chatId);

URL url;

try {

url = new URL(FAVORITES\_PATH);

HttpURLConnection con = (HttpURLConnection) url.openConnection();

con.setRequestMethod("POST");

con.setRequestProperty("X-Api-App-Id", APP\_KEY);

con.setRequestProperty("Authorization", token);

writeData(con.getOutputStream(), "ids[0]=" + vacancyId);

if (con.getResponseCode() == 201) {

String json = readData(con);

return json.contains("true");

} else {

return false;

}

} catch (IOException e) {

e.printStackTrace();

return false;

}

}

@Override

public boolean removeFavoriteVacancy(long chatId, long vacancyId) {

String token = tryGetAuthToken(chatId);

URL url;

try {

url = new URL(FAVORITES\_PATH);

HttpURLConnection con = (HttpURLConnection) url.openConnection();

con.setRequestMethod("POST");

con.setRequestProperty("X-Api-App-Id", APP\_KEY);

con.setRequestProperty("Authorization", token);

writeData(con.getOutputStream(), "ids[0]=" + vacancyId);

return con.getResponseCode() == 204;

} catch (IOException e) {

e.printStackTrace();

return false;

}

}

private HttpURLConnection requestGet(String path, String authToken) {

URL url;

try {

url = new URL(path);

HttpURLConnection con = (HttpURLConnection) url.openConnection();

con.setRequestMethod("GET");

con.setRequestProperty("X-Api-App-Id", APP\_KEY);

if (authToken != null) {

con.setRequestProperty("Authorization", authToken);

}

System.out.println("Response code for path " + path.substring(0, path.lastIndexOf('/')) + " " + con.getResponseCode());

return con;

} catch (IOException e) {

e.printStackTrace();

return null;

}

}

@Override

public UserInfo loadUser(long chatId) {

String token = tryGetAuthToken(chatId);

HttpURLConnection connection = requestGet(CURRENT\_USER\_PATH, token);

UserInfo userInfo = null;

try {

if (connection != null && connection.getResponseCode() == 200) {

String jsonUser = readData(connection);

userInfo = entityMapper.mapUserInfo(jsonUser);

}

} catch (IOException e) {

e.printStackTrace();

}

return userInfo;

}

private String tryGetAuthToken(long chatId) {

AuthService authService = AuthServiceImpl.getInstance();

if (!authService.isLoggedIn(chatId)) {

throw new NotAuthorizedException("User must be authorized to execute operation");

}

AuthToken authToken = authService.getToken(chatId);

return authToken.getTokenType() + " " + authToken.getAccessToken();

}

private String readData(HttpURLConnection connection) {

StringBuilder result = new StringBuilder();

try {

BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));

String line;

while ((line = in.readLine()) != null) {

result.append(line);

}

} catch (IOException e) {

e.printStackTrace();

}

return Objects.requireNonNull(result).toString();

}

private void writeData(OutputStream out, String data) {

try {

out.write(data.getBytes());

} catch (IOException e) {

e.printStackTrace();

}

}

}